

WAFFLE: Multi-Modal Model for Automated Front-End Development

Shanchao Liang
Purdue University
liang422@purdue.edu

Nan Jiang
Purdue University
jiang719@purdue.edu

Shangshu Qian
Purdue University
qian151@purdue.edu

Lin Tan
Purdue University
lintan@purdue.edu

Abstract

Web development involves turning UI designs into functional webpages, which can be difficult for both beginners and experienced developers due to the complexity of HTML's hierarchical structures and styles. While Large Language Models (LLMs) have shown promise in generating source code, two major challenges persist in UI-to-HTML code generation: (1) effectively representing HTML's hierarchical structure for LLMs, and (2) bridging the gap between the visual nature of UI designs and the text-based format of HTML code. To tackle these challenges, we introduce WAFFLE, a new fine-tuning strategy that uses a structure-aware attention mechanism to improve LLMs' understanding of HTML's structure and a contrastive fine-tuning approach to align LLMs' understanding of UI images and HTML code. Models fine-tuned with WAFFLE show up to 9.00 pp (absolute percentage point) higher HTML match, 0.0982 higher CW-SSIM, 32.99 higher CLIP, and 27.12 pp higher LLEM on our new benchmark WebSight-Test and an existing benchmark Design2Code, outperforming current fine-tuning methods.

1 Introduction

While Large Language Models have significantly advanced the automation of code generation in popular programming languages such as Python or Java (Jiang et al., 2023; Touvron et al., 2023; Fried et al., 2023; Nijkamp et al., 2022; Rozière et al., 2024; Guo et al., 2024; Li et al., 2023c; Lozhkov et al., 2024), the automation of HTML code generation from UI design remains under-explored and challenging. As the core of front-end development, this task requires the model to understand not only the transformation from natural languages (NL) to programming languages (PL) but also from visual designs to PL. Recently, Multi-modal Large Language Models (MLLMs) have brought much progress in generating text from image descrip-

<pre> <html> <body style="background-color:rgb(241, 244, 249);"> <div style="text-align:center" style="width:100%; padding:4px;"> <h2 style="color:black"> Selections </h2> <div style="height:80px; background:#d5f0ff; display:flex; align-items:center; justify-content:center;"> Property for Sale </div> </div> <div id="right-column" style="width:100%; padding:5px;"> <h2 style="color:black"> Customer Reviews </h2> <p style="color:#00000099"> I had a great experience at this real estate company. </p> <button> Contact Us </button> </div> </body> </html> </pre> <p>(a) HTML and CSS code</p>	<h2 style="color:blue;">Selections</h2> <p>Property for Sale</p> <h2 style="color:blue;">Customer Reviews</h2> <p>I had a great experience at this real estate company.</p> <p>Contact Us</p> <p>(b) Webpage rendered from code in (a)</p>	<h2 style="color:blue;">Customer Reviews</h2> <p>I had a great experience at this real estate company.</p> <p>Contact Us</p> <p>(c) Webpage rendered after removing code highlighted in yellow from (a)</p>
---	--	---

Figure 1: Removing the children of the element `<div id = "left-column">` highlighted in yellow does not affect the structure of the visual layout of itself or its sibling element `<div id="right-column">`.

tions (Radford et al., 2021; Zhai et al., 2023; Li et al., 2023a; Liu et al., 2023b,a; Li et al., 2022, 2023b; Dai et al., 2023; Blecher et al., 2023; Chen et al., 2023; Wei et al., 2023; vikhvat, 2024). On top of this, a few MLLMs have been fine-tuned using UI image-to-code datasets (e.g., WebSight (Lau-rençon et al., 2024), Design2Code (Si et al., 2024)). Nonetheless, these approaches mainly apply standard fine-tuning and fail to address specific HTML code generation challenges.

Two key challenges exist in translating UI design images to HTML code: (1) how to teach models to learn effectively the **domain knowledge of HTML structures**, which significantly influences the rendered UI design, and (2) how to teach the models to learn the **subtle differences in the visual understanding of UI images and text understanding of HTML code**.

Regarding the first challenge, there are three basic structural aspects of HTML code. Firstly, all the styles of the parent element are directly passed to the children unless specifically overridden. Secondly, the layout of the siblings affects each other. Thirdly, nodes are not affected by the subtrees of their siblings. The last principle might

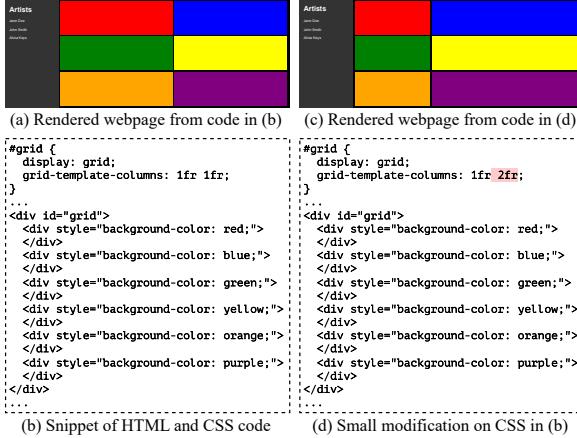


Figure 2: Existing MLLM generates identical HTML and CSS code in (b) given the different webpage screenshots in (a) and (c). The MLLM fails to generate the correct `2fr` highlighted in red in (d).

be less obvious compared to the first two, and we explain it with an example. Figure 1 shows (a) an HTML code file and (b) its rendered webpage. We use blue, orange, and green blocks to map the code chunks and their corresponding visual rendering on the webpage. The top-level `<body>` element refers to the whole webpage, the child element `div id="left-column"` refers to the left part of the webpage, and another child element `div id="right-column"` refers to the right part. Modifications to the child of `div id="left-column"` do not change how the `div id="right-column"` looks on the webpage (even if we remove all the content inside `div id="left-column"` as (c) shows).

To learn such domain knowledge of HTML code structure, we propose a novel **structure-aware attention** mechanism. The structure-aware attention captures the structure information of the HTML code by explicitly allowing tokens to focus on three types of previous code segments that are the most relevant (details in Section 2.2). With such structural information in the HTML code, WAFFLE can focus on parts of the code that have the most influence on the resulting UI design, thus benefiting the overall performance.

Figure 2 illustrates the second challenge, i.e., learning the fine differences and details of the visual input. Figure 2(a) and Figure 2(c) are two highly similar but different UI images of rendered webpages, i.e., the colors and text are identical, but the widths of the columns are slightly different. VLM-WebSight (Laurençon et al., 2024), a state-of-the-art MLLM for webpage image to HTML code generation, fails to capture such small dif-

ferences; thus, it generates identical HTML and CSS code (Figure 2(b)) for the different UI images: Figure 2(a) and Figure 2(c). The model fails to generate `1fr 2fr` (highlighted in code segments (d) with red background) for screenshot (c). In this case, VLM-WebSight’s vision model fails to recognize the visual difference, and its text model is unable to use the encoded visual information to produce accurate textual output.

To enable MLLMs to recognize subtle differences in UI images due to minor changes in the code, we adopt **contrastive learning** (Zhai et al., 2023; Radford et al., 2021; Gao et al., 2021) to the current task to teach MLLMs to focus on important visual differences.

Combining the two approaches, this paper introduces WAFFLE, a fine-tuning pipeline specifically designed for UI images to HTML code generation, with the following contributions:

1. We design *structure-aware attention* for HTML code generation, which enables MLLMs to learn the structure knowledge of HTML code.
2. We apply *contrastive learning* algorithms to boost MLLMs’ understanding of the details in the rendered UI images and the HTML code.
3. We create a new dataset of 231,940 pairs of webpages and their HTML code, which could facilitate future research on web MLLMs.
4. We conduct comprehensive experiments on two backbone MLLMs. WAFFLE improves the backbone MLLMs by achieving up to 9.00 pp higher HTML Match, 0.0982 higher CW-SSIM, 32.99 higher CLIP, and 27.12 pp higher LLEM.
5. We highlight that WAFFLE as a fine-tuning approach, is model-independent and can be applied to improve any MLLMs for UI-to-HTML code generation.

Availability: <https://github.com/lt-asset/Waffle>

2 Approach

Figure 3 represents the overview of WAFFLE. We create a new mutated HTML dataset (Section 2.1) for training and fine-tuning. In addition, we design structure-aware attention (Section 2.2) during model fine-tuning to teach models to focus on important HTML segments. Finally, we use contrastive learning training to teach models to learn visual differences and align the models’ visual and HTML/CSS code understanding (Section 2.3). Specifically, we construct the training dataset by

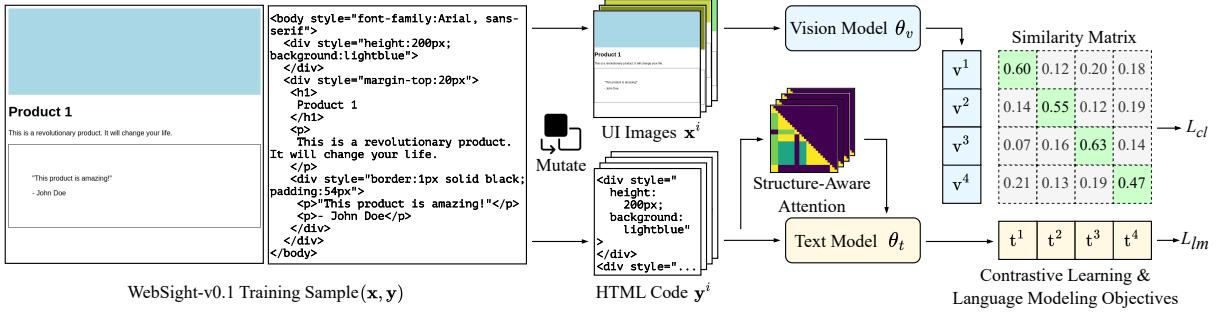


Figure 3: Overview of WAFFLE, including training data mutation, structure-aware attention, and contrastive learning.

Color	Size	Margin	CSS	Font	Display	Position	HTML	Total
12	11	19	10	1	2		8	63

Table 1: Most frequent causes of failures.

applying mutation rules for HTML code on a subset of a popular dataset, WebSight-v0.1, to generate the corresponding source code and UI images.

2.1 Training Data Mutation

To teach MLLMs the important visual differences, we create WAFFLE’s contrastive training data from WebSight-v0.1, which is a fine-tuning dataset built by HuggingfaceM4 and contains 822,987 pairs of HTML code and its corresponding screenshots (Laurençon et al., 2024).

We study the common mistakes of existing web MLLM, VLM-WebSight, which enabled us to create realistic mutation rules to mutate HTML/CSS code in WebSight-v0.1. We conduct a failure analysis of VLM-WebSight on 50 validation data instances. For every mismatched webpage, we manually inspect the root causes of the failures and eventually conclude seven common categories of errors as shown in Table 1. For each category, we create a set of rules to mutate an existing HTML, as shown in Appendix A.1.

Based on the mutation rule, we randomly sample 100,000 instances from the WebSight-v0.1 dataset, creating four distinct mutants with each sample. The mutation rules are applied based on the frequency of failures computed from the validation set. The final mutated dataset (after removing rendering failures, identical mutants, and blank images) has 57,985 groups, each group containing four pairs of HTML code and the corresponding rendered webpage images.

2.2 Structure-Aware Attention

HTML code has clear structures, and certain structural properties can be directly reflected in the ren-

dered UI design. Such domain knowledge can benefit the generation process of MLLMs. There are three most important elements for rendering an element’s layout: its parent element, sibling elements, and the element itself. WAFFLE implements a novel attention mask that provides each element with a pruned view of all previous tokens, including *parent-attention*, *sibling-attention*, and *self-attention*. These attention masks allow the tokens to pay specialized attention to their parent elements, sibling elements, and themselves.

Figure 4 shows a simple example of WAFFLE’s structure-aware attention. Figure 4 (a) shows an HTML code snippet and (b) shows the DOM tree of HTML code in (a), where the root node is the `<body>` element. `<div id="leftCol">` and `<div id="rightCol">` are two children of node `<body>`, and they are siblings to each other. `<div id="leftCol">` has one child, the text `Selections`. `<div id="rightCol">` has one child, which is a `<h2>` element with the text `Customer Reviews` inside.

According to the domain knowledge that an element is mostly affected by its parent and sibling elements, WAFFLE builds the structure-aware attention as shown in (c).

Parent-Attention. The parent-attention is from each element’s tokens to its parent element’s tokens. WAFFLE utilizes the fact that all the children elements inherit the parent element’s styles and structure. For instance, the tokens of the element `<div id="leftCol">` pay parent-attention to tokens of the element `<body>`, and the tokens of the element `Selections` pay parent-attention to tokens of the element `<div id="leftCol">`.

Sibling-Attention. The sibling attention is from each element’s tokens to its preceding sibling elements’ tokens. WAFFLE utilizes the fact that sibling elements under the same parent can affect the arrangement and style of each other, so each element needs to pay attention to its preceding sib-

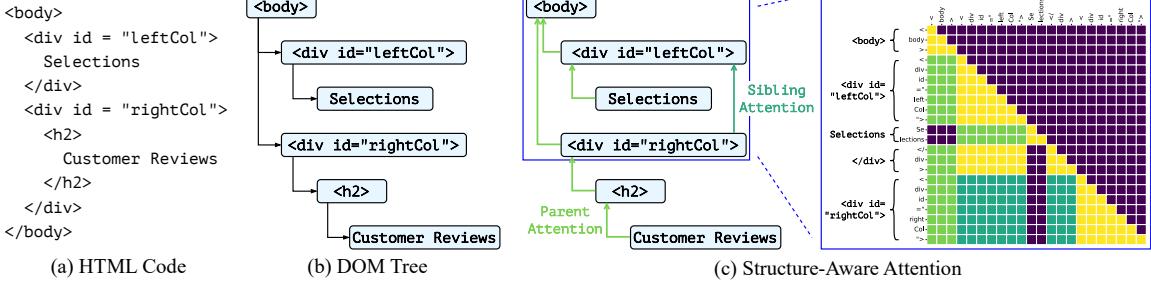


Figure 4: Example of structure-aware attention.

lings. For instance, the tokens of the element `<div id="rightCol">` pay sibling-attention to tokens of the element `<div id="leftCol">`.

Self-Attention. This is the standard self-attention mechanism, it allows each token to focus on all previous tokens within the same element, excluding the children elements. To illustrate, all tokens in a specific element have self-attention (yellow cells Figure 4) to all the tokens belonging to the element itself.

WAFFLE applies the structure-aware attention to the language model decoder only, and keeps the vision encoder as is. WAFFLE only applies the structure-aware attention mechanism to one-fourth of the attention heads in the language model decoder. This enables a quarter of attention heads to learn the structural domain knowledge explicitly, and the other three-quarters of heads to keep the standard full self-attention and pre-training knowledge. The number of attention heads that use structure-aware attention is a hyper-parameter, which can be tuned as in Section 4.2.

2.3 Contrastive-Learning

The example illustrated in Figure 2 highlights a critical gap in current models: their limited ability to effectively map variations in HTML code to the corresponding webpage screenshot. To address this, WAFFLE utilizes contrastive learning, allowing models to learn from comparisons and contrasts between similar examples.

More concretely, the training dataset consists of groups of HTML codes and UI Image pairs, where each group has k such pairs. During training, in any group, for image \mathbf{x}^i and code \mathbf{y}^i ($i \in \{1, \dots, k\}$), image \mathbf{x}^i is the rendered webpage image from code \mathbf{y}^i , the webpage image is split into a list of $P \times P$ pixels patches (P depends on the configuration of the backbone MLLM), each patch is encoded by the vision model θ_v and fused to the text model’s latent space using an adapter to result in a

list of P patch embeddings $\mathbf{v}^i = \{v_1^i, v_2^i, \dots, v_M^i\}$ (M is a hyper-parameter of the backbone MLLM). The HTML code is tokenized into tokens $\mathbf{y}^i = \{y_1^i, y_2^i, \dots, y_{N^i}^i\}$, where N^i is the number of the tokens of the HTML code \mathbf{y}^i . These tokens are encoded to embeddings $\mathbf{t}^i = \{t_1^i, t_2^i, \dots, t_{N^i}^i\}$ by the language model θ_t (using the structure-aware attention). We use the average over all the patch embeddings to represent the embeddings of the webpage image, and the average overall all the tokens’ embeddings to represent the embedding of the whole HTML code, denoted by:

$$\bar{v}^i = \frac{1}{P} \sum_{j=1}^P v_j^i, \quad \bar{t}^i = \frac{1}{N^i} \sum_{j=1}^{N^i} t_j^i \quad (1)$$

Then we calculate the cosine similarity score between the text embeddings and image embeddings, \bar{t}^i and \bar{v}^i , bipartitely. The contrastive learning objective is to maximize the similarity between the embeddings of an HTML code and that of the corresponding UI image, which is achieved by minimizing the contrastive learning loss L_{cl} , along with the standard language modeling fine-tuning loss L_{lm} as follows:

$$L_{cl} = - \sum_{i=1}^k \left(\frac{\exp(\text{sim}(\bar{t}^i, \bar{v}^i))}{\sum_{j=1}^k \exp(\text{sim}(\bar{t}^i, \bar{v}^j))} \right) \quad (2)$$

$$L_{lm} = - \sum_{i=1}^k \sum_{j=1}^{N^i} \log P(y_j^i | \mathbf{y}_{<j}^i, \mathbf{x}^i, \theta_t, \theta_v) \quad (3)$$

The contrastive learning loss aims to maximize the similarity scores at the diagonal of the similarity matrix (as the green cells of the similarity matrix shown in Figure 3). It trains the MLLM’s vision model, θ_v , which encodes a webpage, \mathbf{x}^i , to closely match the encoded embeddings of the text model, θ_t , on the corresponding HTML code, \mathbf{y}^i .

The language modeling loss, on the other hand, aims to maximize the probability of generating the correct token y_j^i given all previous token $\mathbf{y}_{<j}^i$ and the input webpage image \mathbf{x}^i . This is the standard

objective of training an MLLM to generate text from images. WAFFLE jointly optimizes the two objectives as follows:

$$L_{\text{WAFFLE}} = L_{lm} + \lambda L_{cl} \quad (4)$$

where λ is a hyper-parameter constant controlling the effect of contrastive learning on the overall optimization.

3 Experimental Setup

3.1 Model Training

We implement WAFFLE on two backbones, VLM-WebSight, and Moondream2([Laurençon et al., 2024; vikhyat, 2024](#)). Each backbone is first fine-tuned on the WebSight-v0.1 dataset using the standard language modeling objective. For VLM-Websight, we use the released fine-tuned checkpoint with details in ([Laurençon et al., 2024](#)). This checkpoint is fine-tuned using DoRA ([Liu et al., 2024](#)) (a variant of parameter-efficient training, LoRA ([Hu et al., 2022](#)), with rank set to 64). For Moondream2, we also fine-tune the model using DoRA ([Liu et al., 2024](#)) (with rank set to 64, and lora_alpha set to 128). The model’s weights are updated using the AdamW ([Loshchilov and Hutter, 2019](#)) optimizer, with the learning rate set to $3e^{-5}$. The batch size is 64.

On top of the fine-tuned MLLMs from the first step, we apply the structure-aware attention and contrastive learning approach. Structure-aware attention is applied to $\frac{1}{4}$ of the attention heads in each attention layer in the LLM decoder. Each model is trained with DoRA using the combined learning objective on the contrastive learning dataset (Equation (4) with λ set to 0.1). The model’s weights are updated using the AdamW optimizer, with the learning rate set to $2e^{-5}$. The batch size is 32.

3.2 Test Data

We evaluate WAFFLE using two test datasets: WebSight-Test, which consists of synthetic webpages, and Design2Code, which consists of real-world webpages. Since WebSight-v0.1 was already used for training, we created WebSight-Test following the same process as WebSight-v0.1 ([Laurençon et al., 2024](#)). In total, WebSight-Test contains 500 test samples, each has a webpage image and the respective ground-truth HTML source code.

Design2Code is an open-source benchmark of 484 manually processed real-world website screenshots, which are much more complex than

Websight-v0.1. Evaluation on Design2Code indicates the generalization ability of models fine-tuned on WAFFLE’s training dataset to real-world scenarios ([Si et al., 2024](#)).

3.3 Evaluation Metrics

HTML-Match. HTML-Match is the percentage of generated images that match the ground truth images perfectly at the pixel level. For this comparison, styles and attributes are removed from both the ground truth and generated HTML. This process emphasizes the model’s ability to accurately recognize the text content and the DOM tree structure of the HTML code.

CLIP. CLIP score ([Radford et al., 2021; Si et al., 2024](#)) measures the similarity between the rendered webpage of the inferred HTML code and the ground truth webpage based on the CLIP-ViT-B/32’s embeddings of webpages.

Low-Level Element Matching (LLEM). Previous work ([Si et al., 2024](#)) proposes LLEM to measure the percentage of matched (1) text blocks, (2) text content, (3) position of each matched text block, and (4) font color within each text block.

CW-SSIM. The complex wavelet structural similarity index (CW-SSIM) computes the structural similarity between images ([Sampat et al., 2009](#)).

Human Evaluation. Two human annotators are asked to compare the rendered webpages generated by different techniques from a subset of test data with the ground-truth webpage and rank them.

4 Results

4.1 Effectiveness of WAFFLE

Table 2 and Table 3 show the performance of various fine-tuning strategies on the two testing datasets, WebSight-Test (500 samples) and Design2Code (484 samples).

Comparison against standard fine-tuning. We compare WAFFLE with the baseline, the standard fine-tuning (FT) method. In both tables, WAFFLE achieves significant improvements over the standard FT on *all metrics* with Moondream2 and VLM-WebSight as the backbone.

On WebSight-Test, WAFFLE achieves 6.00 pp higher HTML-Match (27.60% vs. 21.60%) and 0.0253 higher CW-SSIM (0.4486 vs. 0.4233) than Standard FT with Moondream2 as the backbone. On the VLM-WebSight backbone, WAFFLE outperforms Standard FT by 9.00 pp HTML-Match,

Backbones	Techniques	HTML-Match (%) ↑	CW-SSIM ↑	CLIP ↑	Low-Level Element Matching (LLEM) (%) ↑				
					Average	Block-Match	Text	Position	Color
Gemini 1.5 Pro	Prompting	9.40	0.3385	88.55	90.16	94.31	98.41	84.73	83.18
GPT-4o mini	Prompting	10.20	0.3055	87.72	87.54	92.59	98.48	82.65	76.45
GPT-4o	Prompting	11.40	0.3666	89.03	92.18	94.66	98.43	87.04	88.60
Moondream2	Standard FT	21.60	0.4233	89.92	90.59	91.73	96.98	87.56	86.77
	WAFFLE	27.60	0.4486	89.98	91.72	92.26	97.25	89.55	87.81
VLM-WebSight	Standard FT	28.00	0.5023	93.30	92.73	97.95	90.72	91.07	93.45
	WAFFLE	37.00	0.6005	94.57	95.16	93.62	98.16	93.29	95.57

Table 2: Main results on the WebSight-Test dataset.

Backbones	Techniques	CW-SSIM ↑	CLIP ↑	Low-Level Element Matching (LLEM) (%) ↑				
				Average	Block-Match	Text	Position	Color
Gemini 1.5 Pro*	Prompting	0.2652	87.76	87.17	91.82	97.40	82.67	76.81
GPT-4o-mini	Prompting	0.2304	86.06	78.84	70.64	92.39	78.55	73.78
GPT-4o	Prompting	0.2776	89.03	83.67	75.98	94.29	83.38	81.01
Moondream2	Standard FT	0.1348	46.63	40.71	29.56	49.41	40.73	43.14
	WAFFLE	0.2142	79.62	67.83	44.32	83.59	71.61	71.81
VLM-WebSight	Standard FT	0.2518	82.35	73.00	55.77	84.14	74.74	77.36
	WAFFLE	0.2815	85.98	77.81	61.47	88.20	79.30	82.28

*Gemini 1.5 Pro’s results are on 384 test instances as it generates no answers for the remaining 100 instances.

Table 3: Main results on the Design2Code dataset.

0.0982 CW-SSIM, and 2.43% averaged LLEM. On Design2Code, WAFFLE achieves greater improvements compared to the Standard FT technique with both backbones.

Note that, WAFFLE is a generalizable fine-tuning pipeline that can benefit any pre-trained MLLMs.

Summary: Overall, WAFFLE significantly improves all metrics for both backbones in both test datasets over standard fine-tuning with up to 9.00 pp for HTML Match, 0.0982 for CW-SSIM, 32.99 for CLIP, and 27.12 pp for LLEM.

Comparison against SOTA commercial models. Due to the lack of comparable baselines, we also compare WAFFLE with top commercial models, which include GPT-4o mini, GPT-4o, and Gemini 1.5 Pro. We apply the direct prompting method following prior work (Si et al., 2024). The result is shown in Table 2.

On the WebSight-Test dataset, models fine-tuned with WAFFLE perform better than the SOTA commercial models. VLM-WebSight overperforms GPT-4o by 25.60 pp on HTML-Match (37.00% vs. 11.40%) and 0.2339 on CW-SSIM (0.6005 vs. 0.3666). Similarly, for the smaller backbone, Moondream2 exceeds GPT-4o by 16.2 pp on HTML-Match (27.60% vs. 11.40%) and 0.0820 on CW-SSIM (0.4486 vs. 0.3666).

As shown in Table 3, VLM-WebSight fine-tuned by WAFFLE is better than GPT-4o on CW-SSIM by 0.039, and better than GPT-4o mini by 0.511

on the Design2Code dataset. However, GPT-4o is better in the other two metrics versus VLM-WebSight. Moondream2 fine-tuned by WAFFLE has a lower performance compared to GPT-4o and GPT-4o mini on all metrics. This is likely due to its smaller size, which could influence its generalizability to more complex, out-of-distribution data on the Design2Code dataset. Case studies are provided in Appendix A.2.

Summary: On simpler data, WAFFLE achieves better or comparable results than SOTA commercial models, with 16.20–25.60 pp improvement on HTML-Match and 0.0820–0.2339 improvement on CW-SSIM. On more complex data, WAFFLE enables VLM-WebSight to outperform commercial models on CW-SSIM.

4.2 Ablation Studies

Comparison against ablation models. We compare WAFFLE with two ablation models:

- WAFFLE_{-attn}: This is WAFFLE with only contrastive learning and without the use of structure-aware attention.
- WAFFLE_{-contra}: This is WAFFLE with only structure-aware attention.

Table 4 shows the ablation study results. WAFFLE_{-attn} brings improvements compared to the Standard FT on all metrics, and WAFFLE_{-contra} brings a 4.40 pp improvement on HTML-Match. On the Design2Code dataset, WAFFLE has domi-

Backbones	Techniques	WebSight-Test				Design2Code		
		HTML-Match (%) \uparrow	CW-SSIM \uparrow	CLIP \uparrow	LLEM (%) \uparrow	CW-SSIM \uparrow	CLIP \uparrow	LLEM (%) \uparrow
Moondream2	Standard FT	21.60	0.4233	89.92	90.59	0.1348	46.63	40.71
	WAFFLE- _{attn}	23.60	0.4311	90.47	91.34	0.1821	67.73	56.49
	WAFFLE- _{contra}	26.00	0.4296	89.55	91.21	0.2100	76.63	65.82
	WAFFLE	27.60	0.4486	89.98	91.72	0.2142	79.62	67.83
VLM-WebSight	Standard FT	28.00	0.5023	93.30	92.73	0.2518	82.35	73.00
	WAFFLE- _{attn}	30.80	0.5411	94.29	94.20	0.2480	85.64	75.34
	WAFFLE- _{contra}	35.80	0.5677	95.08	95.30	0.2653	85.16	76.48
	WAFFLE	37.00	0.6005	94.57	95.16	0.2815	85.98	77.81

Table 4: Ablation studies on the two test datasets. LLEM refers to the averaged Low-Level Element Matching.

Techniques	Rank 1 \uparrow	Rank 2 \uparrow	Rank 3 \uparrow	Avg Rankings \downarrow
Standard FT	7 17 (24)	14 13 (27)	17 18 (35)	2.90 2.42 (2.66)
WAFFLE- _{attn}	15 16 (31)	9 17 (26)	24 16 (40)	2.55 2.37 (2.46)
WAFFLE- _{contra}	38 20 (58)	8 11 (19)	10 15 (25)	1.67 2.38 (2.02)
WAFFLE	27 32 (59)	18 12 (30)	10 9 (19)	1.88 1.85 (1.87)

Table 5: Human evaluation on two datasets using VLM-WebSight as the backbone. The numbers are shown as “x/y (x+y)”, where x is the result on WebSight-Test and y is the result on Design2Code.

nating performance on *all metrics* for models fine-tuned with both backbones. Across the two backbones, models fine-tuned with WAFFLE are higher than those fine-tuned with WAFFLE-_{attn} by up to 0.0335 on CW-SSIM.

Summary: Contrastive learning and structure-aware attention significantly improve performance over standard fine-tuning. On the simpler WebSight-Test data, models trained with WAFFLE achieve the highest HTML-Match and CW-SSIM scores. On the more complex Design2Code data, WAFFLE still delivers the best results across all metrics for both backbones.

Human evaluation results. We select 30 test samples from WebSight-Test and Design2Code (60 total). Each sample has four generated HTML codes and rendered webpages from Standard FT, WAFFLE-_{attn}, WAFFLE-_{contra}, and WAFFLE. Human raters rank the generated webpages based on similarity to the ground-truth webpage without knowing which model produced each one. Table 5 shows the human evaluation results for VLM-WebSight fine-tuned by Standard FT, WAFFLE-_{attn}, WAFFLE-_{contra}, and WAFFLE.

Across both datasets, WAFFLE has the best averaged-rankings, 1.87, outperforming both ablations and the baseline. Specifically, WAFFLE reaches 32 times rank 1 placement on Design2Code, showing great generalizability on more complex datasets. WAFFLE-_{contra} is the second best technique on the two testsets, reaching 58 times rank 1, and an average ranking of 2.02. On

Techniques	Prior	Current	Drop (%)
WAFFLE- _{attn}	0.8002	0.5797	27.55
WAFFLE	0.8291	0.7932	4.34

Table 6: CW-SSIM on 20 samples using the VLM-WebSight backbone. “Prior” refers to “without intermediate mistakes”, and “Current” to “with intermediate mistakes”.

the other hand, WAFFLE-_{attn} is the third-best technique but still outperforms standard FT.

Summary: Human evaluation shows that (1) both structure-aware attention and contrastive learning contribute to the code generation quality, and (2) WAFFLE-generated HTML/CSS code is consistently rated higher than code generated with standard fine-tuning.

4.3 Structure-Aware Attention’s Effect

To demonstrate how structure-aware attention helps MLLMs focus on the correct elements (like parent and sibling elements) during generation, we simulate 20 cases where generation errors occur mid-process. Ideally, mistakes within sibling elements should not disrupt the generation of the main element. We select 20 high-performing samples generated by VLM-WebSight fine-tuned with WAFFLE and WAFFLE-_{attn}. Using the HTML code generated by the models, we manually crop and mutate sections to simulate errors. The models are then tasked with re-completing the HTML code starting from the error. Our goal is to evaluate whether the models are robust enough to handle intermediate errors without causing major failures in subsequent generations.

Table 6 shows the results of re-completion following the intermediate mistakes. With WAFFLE-_{attn}, the averaged CW-SSIM across the 20 samples drops by 0.2205 (from 0.8002 to 0.5797) if the model makes intermediate mistakes. By contrast, with WAFFLE, the averaged CW-SSIM only drops by 0.0359, from 0.8291 to 0.7932.

Summary: Integrating structure-aware attention brings great stability to model generation, making models more robust against intermediate mistakes, and reducing the performance drop by 23.31%, from 27.55% to 4.24%.

5 Related Work

5.1 Multi-Model Large Language Models

Recent advances in vision and language models have greatly improved MLLMs’ capabilities in tasks like image captioning (Radford et al., 2021; Zhai et al., 2023; Li et al., 2022, 2023b; McKinzie et al., 2024; Alayrac et al., 2022; Laurençon et al., 2023), text-to-image generation (Ramesh et al., 2022; Rombach et al., 2021), and visual question answering (Dai et al., 2023; Liu et al., 2023b,a; Bai et al., 2023). While popular models like Llava (Liu et al., 2023b,a), Qwen-VL (Bai et al., 2023), and Vary (Wei et al., 2023) perform well in general image tasks, they don’t focus on converting UI images to HTML code. To address this, we propose WAFFLE, a fine-tuning method that equips MLLMs with the domain-specific knowledge needed for UI-to-HTML generation.

5.2 Attention Mechanism

The attention mechanism is the key part of modern Transformer (Vaswani et al., 2017) architectures, as it effectively captures the hidden features of input data. To handle the challenges of certain domains, specialized attention mechanisms have been explored, such as pyramid attention (Chai and Li, 2022) and hierarchical attention (Guo et al., 2023; Shi et al., 2021; Nguyen et al., 2023; Yang et al., 2016) designed for long-range, cross-file code understanding and generation, as well as regularized attention for assembly code (Su et al., 2024). Different from existing work, WAFFLE targets HTML code, with the new challenge of its restricted structure. WAFFLE designs a novel structure-aware attention to learn such structure knowledge.

5.3 UI to HTML Generation

Early direction for UI code generation utilizes sketch webpage figures, e.g., hand-drawn website sketches, to generate UI code that can be rendered into similar images as sketch images (Robinson, 2019). Yet, this direction is not practical, as not all front-end developers want to draw out a sketch website when they need help from an automated tool. In contrast, leveraging advancements

in MLLMs, Huggingface has recently released WebSight, which is trained on the WebSight-v0.1 dataset (Laurençon et al., 2024). Although specific details of the model are not disclosed, it represents a significant shift towards end-to-end UI to code generation. Similarly, Design2Code-18B is a model using CogAgent as the backbone using a subset of WebSight-v0.1 (Si et al., 2024; Hong et al., 2023). However, neither WebSight nor Design2Code tries to adapt domain knowledge of HTML for this task. In contrast, we provide structure-aware attention and apply contrastive learning with the mutations to teach the model the fine-grained difference of HTML images.

6 Limitation

One limitation of WAFFLE is that it has only been implemented on two models: VLM-WebSight and Moondream2. While WAFFLE could potentially be applied to any MLLM, further exploration is limited by computing resources. Our experiments show that WAFFLE brings significant improvements over standard fine-tuning on these two models, indicating some level of generalizability. Another limitation is that the metrics we use do not fully capture human evaluation. HTML-Match overlooks CSS styling, and metrics like CW-SSIM, CLIP, and LLEM are similarity-based, which can lead to unreliable scores. Evaluating HTML code automatically is challenging, so we include CLIP and LLEM, as used in previous work (Si et al., 2024; Gui et al., 2024), along with CW-SSIM and HTML-Match to ensure fair evaluation.

7 Conclusion

This work presents WAFFLE, a fine-tuning pipeline for UI-to-HTML code generation. WAFFLE introduces a structure-aware attention mechanism to capture HTML structure and employs contrastive learning to align visual and textual understanding, aiding MLLMs in distinguishing subtle webpage differences. WAFFLE outperforms standard fine-tuning on two backbone MLLMs, with improvements of up to 9 pp in HTML Match, 0.0982 in CW-SSIM, 32.99 in CLIP, and 27.12 pp in LLEM. Ablation studies confirm that both key components of WAFFLE contribute to better cross-modality understanding and more robust code generation. Notably, WAFFLE is model-independent and can enhance any MLLMs for UI-to-HTML code generation.

References

- Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. 2022. **Flamingo: a visual language model for few-shot learning.** *Preprint*, arXiv:2204.14198.
- Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. **Qwen-vl: A frontier large vision-language model with versatile abilities.** *arXiv preprint arXiv:2308.12966*.
- Lukas Blecher, Guillem Cucurull, Thomas Scialom, and Robert Stojnic. 2023. **Nougat: Neural optical understanding for academic documents.** *Preprint*, arXiv:2308.13418.
- Lei Chai and Ming Li. 2022. Pyramid attention for source code summarization. *Advances in Neural Information Processing Systems*, 35:20421–20433.
- Lin Chen, Jisong Li, Xiaoyi Dong, Pan Zhang, Conghui He, Jiaqi Wang, Feng Zhao, and Dahu Lin. 2023. **Sharegpt4v: Improving large multimodal models with better captions.** *arXiv preprint arXiv:2311.12793*.
- Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. 2023. **Instructclip: Towards general-purpose vision-language models with instruction tuning.** *Preprint*, arXiv:2305.06500.
- Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen tau Yih, Luke Zettlemoyer, and Mike Lewis. 2023. **Incoder: A generative model for code infilling and synthesis.** *Preprint*, arXiv:2204.05999.
- Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. **SimCSE: Simple contrastive learning of sentence embeddings.** In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.
- Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Yi Su, Shaoling Dong, Xing Zhou, and Wenbin Jiang. 2024. **Vision2ui: A real-world dataset with layout for code generation from ui designs.** *Preprint*, arXiv:2404.06369.
- Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. 2023. Longcoder: a long-range pre-trained language model for code completion. In *Proceedings of the 40th International Conference on Machine Learning*, ICML’23. JMLR.org.
- Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. **Deepseek-coder: When the large language model meets programming – the rise of code intelligence.** *Preprint*, arXiv:2401.14196.
- Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. 2023. **Cogagent: A visual language model for gui agents.** *Preprint*, arXiv:2312.08914.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. **LoRA: Low-rank adaptation of large language models.** In *International Conference on Learning Representations*.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. **Mistral 7b.** *Preprint*, arXiv:2310.06825.
- Hugo Laurençon, Lucile Saulnier, Léo Tronchon, Stas Bekman, Amanpreet Singh, Anton Lozhkov, Thomas Wang, Siddharth Karamcheti, Alexander M. Rush, Douwe Kiela, Matthieu Cord, and Victor Sanh. 2023. **Obelics: An open web-scale filtered dataset of interleaved image-text documents.** *Preprint*, arXiv:2306.16527.
- Hugo Laurençon, Léo Tronchon, and Victor Sanh. 2024. **Unlocking the conversion of web screenshots into html code with the websight dataset.** *Preprint*, arXiv:2403.09029.
- Dongxu Li, Junnan Li, Hung Le, Guangsen Wang, Silvio Savarese, and Steven C.H. Hoi. 2023a. **LAVIS: A one-stop library for language-vision intelligence.** In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 31–41, Toronto, Canada. Association for Computational Linguistics.
- Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023b. **Blip-2: bootstrapping language-image pre-training with frozen image encoders and large language models.** In *Proceedings of the 40th International Conference on Machine Learning*, ICML’23. JMLR.org.
- Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. 2022. **BLIP: Bootstrapping language-image pre-training for unified vision-language understanding and generation.** In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 12888–12900. PMLR.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokolov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023c. **Starcoder: may the source be with you!**. *Preprint*, arXiv:2305.06161.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2023a. Improved baselines with visual instruction tuning.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023b. **Visual instruction tuning**. *Preprint*, arXiv:2304.08485.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. **DoRA: Weight-decomposed low-rank adaptation**.

Ilya Loshchilov and Frank Hutter. 2019. **Decoupled weight decay regularization**. *Preprint*, arXiv:1711.05101.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokolov, Christopher Akiki, Marc Marone, Cheng-hao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sébastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. **Starcoder 2 and the stack v2: The next generation**. *Preprint*, arXiv:2402.19173.

Brandon McKinzie, Zhe Gan, Jean-Philippe Fauconnier, Sam Dodge, Bowen Zhang, Philipp Dufter, Dhruti Shah, Xianzhi Du, Futang Peng, Floris Weers, Anton Belyi, Haotian Zhang, Karanjeet Singh, Doug Kang, Ankur Jain, Hongyu Hè, Max Schwarzer, Tom Gunter, Xiang Kong, Aonan Zhang, Jianyu Wang, Chong Wang, Nan Du, Tao Lei, Sam Wiseman, Guoli Yin, Mark Lee, Zirui Wang, Ruoming Pang, Peter Grasch, Alexander Toshev, and Yinfei Yang. 2024. **Mml1: Methods, analysis & insights from multimodal lilm pre-training**. *Preprint*, arXiv:2403.09611.

Minh Huynh Nguyen, Nghi D. Q. Bui, Truong Son Hy, Long Tran-Thanh, and Tien N. Nguyen. 2023. **Hierarchynet: Learning to summarize source code with heterogeneous representations**. *Preprint*, arXiv:2205.15479.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. A conversational paradigm for program synthesis. *arXiv preprint*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. **Learning transferable visual models from natural language supervision**. In *International Conference on Machine Learning*.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. **Hierarchical text-conditional image generation with clip latents**. *Preprint*, arXiv:2204.06125.

Alex Robinson. 2019. **Sketch2code: Generating a website from a paper mockup**. *Preprint*, arXiv:1905.13750.

Robin Rombach, A. Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2021. **High-resolution image synthesis with latent diffusion models**. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10674–10685.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémie Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. **Code llama: Open foundation models for code**. *Preprint*, arXiv:2308.12950.

Mehul P. Sampat, Zhou Wang, Shalini Gupta, Alan Conrad Bovik, and Mia K. Markey. 2009. **Complex wavelet structural similarity: A new image similarity index**. *IEEE Transactions on Image Processing*, 18(11):2385–2401.

Ensheng Shi, Yanlin Wang, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2021.

CAST: Enhancing code summarization with hierarchical splitting and reconstruction of abstract syntax trees. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4053–4062, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2024. **Design2code: How far are we from automating front-end engineering?** *Preprint*, arXiv:2403.03163.

Zian Su, Xiangzhe Xu, Ziyang Huang, Zhuo Zhang, Yapeng Ye, Jianjun Huang, and Xiangyu Zhang. 2024. **Codeart: Better code models by attention regularization when symbols are lacking.** *Proc. ACM Softw. Eng.*, 1(FSE).

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambo, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. **Llama: Open and efficient foundation language models.** *Preprint*, arXiv:2302.13971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. **Attention is all you need.** In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

vikhyat. 2024. **Moondream: tiny vision language model.**

Haoran Wei, Lingyu Kong, Jinyue Chen, Liang Zhao, Zheng Ge, Jinrong Yang, Jianjian Sun, Chunrui Han, and Xiangyu Zhang. 2023. **Vary: Scaling up the vision vocabulary for large vision-language models.** *arXiv preprint arXiv:2312.06109*.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. **Hierarchical attention networks for document classification.** In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California. Association for Computational Linguistics.

X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer. 2023. **Sigmoid loss for language image pre-training.** In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11941–11952, Los Alamitos, CA, USA. IEEE Computer Society.

A Appendix

A.1 Mutation Rules

Table 7 shows the mutation rules we used to mutate the HTML code and create the contrastive learning dataset. Both HTML code and the CSS styles for

each element are mutated according to the failure types in our manual analysis

For CSS styles, we mutate the properties of 1) color, 2) size, 3) margin, 4) font size, 5) type of the element bounding box (display), and 6) positioning of each element. Column “Specification” in Table 7 shows the details of the valid values for each property. For HTML codes, we randomly duplicate one HTML element excluding the ones that will cause render failures (i.e., `<head>`, `<header>`, `<html>`, `<body>`).

Class	Failure Type	Specification
CSS	Color	Random Color in Range [#000000, #FFFFFF]
	Size	Random Size in [0, 500] pixels
	Margin	Random Size in [0, 100] pixels
	Font	Random Size in [0, 40] pixels
	Display	Random Keyword for <code>text-align</code> , <code>display</code> , <code>flex-direction</code> , and <code>justify-content</code>
	Position	Random Keyword for <code>border-radius</code> , <code>position</code> , <code>top</code> , and <code>right</code>
HTML	Structure	Duplication of a Random HTML Element, excluding <code><head></code> , <code><header></code> , <code><html></code> , <code><body></code>

Table 7: Specification for Mutation Rules to construct the Contrastive dataset.

A.2 Case Study

Figure 5 shows the generation results for one instance from WebSight-Test. The generated webpage of GPT-4o has a CW-SSIM of 0.1353, significantly lower than that of 0.3760 from VLM-WebSight under standard fine-tuning. On the other hand, the webpage generated by VLM-WebSight fine-tuned by WAFFLE reaches an almost perfect CW-SSIM score, 0.9995. This example shows the effectiveness of using WAFFLE on the UI image to HTML code task.

A.3 Tuning the Integration of WAFFLE’s Structure-Aware Attention.

WAFFLE applies the structure-aware attention on the attention layer in MLLM’s decoder. To study the portion of attention heads that use structure-aware attention, we fine-tuned VLM-WebSight on a subset of our training dataset (40,000 pairs of HTML code and webpage screenshots). We set the portion of attention heads using structure-aware attention from $\frac{1}{8}$ to all, incrementing each setting by $\frac{1}{8}$. All models are trained with a batch size of 4, and a learning rate of $2e^{-5}$. The models are then evaluated on the validation dataset. We use two metrics to decide the final hyper-parameter: averaged LLEM score and training loss.

Figure 6 (a) shows the averaged LLEM score.



Figure 5: Example test instance from WebSight-Test dataset, with the generated images by GPT-4o, Standard FT, and WAFFLE.

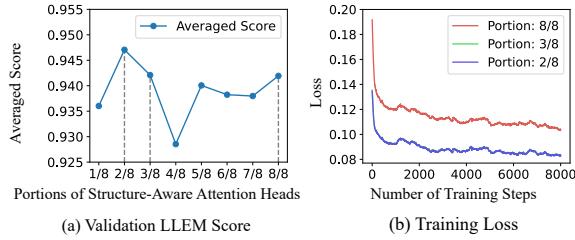


Figure 6: Illustration of the tuning process of the parameter that controls the effect of structure-aware attention. In (b), the green line almost overlaps with the blue line.

Applying structure-aware attention on $\frac{2}{8}$, $\frac{3}{8}$, and $\frac{8}{8}$ of the attention heads results in the top three validation scores. We also consider their training loss in Figure 6 (b). Although applying structure-aware attention on all (i.e., $\frac{8}{8}$) of the attention heads yields a high LLEM score, it also results in a high training loss, likely due to the regular attention heads retaining some prior knowledge during pre-training. In contrast, $\frac{2}{8}$, $\frac{3}{8}$ show similar and lower training losses. Combining these results, we select $\frac{2}{8}$ (i.e., $\frac{1}{4}$) as the final hyper-parameter controlling the portion of attention heads that use structure-aware attention.

A.4 Contrastive Learning’s Effect

To show contrastive learning’s effect on MLLMs’ visual and textual understanding, we design two experiments. The first experiment analyzes whether MLLM’s understanding of the image and code is aligned through the integration of contrastive learn-

Techniques	$d(\bar{v}^i, \bar{t}^i) \downarrow$	$sim(\bar{v}^i, \bar{t}^i) \uparrow$
Standard FT	1.3395	0.1027
WAFFLE-attn	0.8447	0.6244

Table 8: Distance (d) and similarity (sim) between averaged image embeddings \bar{v}^i and text embeddings \bar{t}^i , using Moondream2 as the backbone.

ing, and the second experiment analyzes whether contrastive learning can teach the model to capture the subtle difference in the images.

Aligning models’ two modalities. Specifically, under the same procedure, we compute the averaged text embeddings and image embeddings for a subset (12 samples from WebSight-Test dataset) of the test samples in Section 4.3 using the Moondream2 model fine-tuned by WAFFLE-attn and Standard FT. Then for each pair of the averaged image embeddings and text embeddings, (\bar{v}^i, \bar{t}^i) , we normalize them and compute the Euclidean distance and the cosine similarity between them.

Table 8 shows the results of the measurements for both techniques. The Euclidean distance between the embeddings of the two modalities is 0.8447 for WAFFLE-attn, which is lower than 1.3395, the distance of the embeddings from Standard FT, by 0.4798. Similarly, the cosine similarity between the embeddings encoded by WAFFLE-attn is higher than the Standard FT by 0.5017 (0.6244 vs. 0.1027).

In addition, Figure 7 demonstrates that contrastive learning teaches the model to align the text

Techniques	$d(\bar{v^i}, c) \uparrow$	$sim(\bar{v^i}, c_g) \downarrow$
Standard FT	0.1224	0.9910
WAFFLE _{-attn}	0.7590	0.6202

Table 9: Distance (d) and similarity (sim) between each averaged image embeddings $\bar{v^i}$ with the corresponding centroid c of the group of mutants, with Moondream2 backbone.

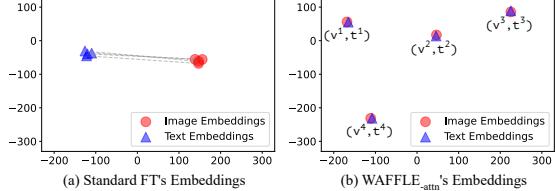


Figure 7: t-SNE plots of the text and image embeddings, computed by Moondream2 fine-tuned with Standard FT and WAFFLE_{-attn}.

and image understandings. The vision embeddings (red circles) are far away from their corresponding text embeddings (blue triangles) when calculated by Standard FT. By contrast, the vision embeddings are grouped with their corresponding text embeddings by WAFFLE_{-attn}.

Capturing subtle visual differences. Using the same computed embeddings, we compute the averaged distances and similarities between each image embedding the centroid of its corresponding group of mutants. Formally, for each group of mutants, G , consisting of image embedding $\{\bar{v^i}\}$, $v^i \in G$, the centroid of the image embeddings is computed as: $c = \sum_{i=1}^{|G|} \bar{v^i}$. Table 9 shows the distance and cosine similarities between the image embeddings. The average distance between each image embedding with its respective centroid computed by the WAFFLE_{-attn} is 0.7590, greatly surpassing the average distance of 0.1224 computed by Standard FT. Likewise, the cosine similarity between the image embeddings is much lower for WAFFLE_{-attn} (0.6202 vs. 0.9910), showing WAFFLE_{-attn}'s better ability to distinguish between the images.

Figure 7 also shows that Standard FT encodes the four different images almost the same in the latent space (i.e., the four red circles are overlapped in (a)), while WAFFLE_{-attn} is able to encode them differently.

A.5 Infrastructure

Our approach is implemented with the following packages: transformers 4.41.1, pytorch 2.3.0, selenium, deepspeed 0.14.1, accelerate 0.30.1, and

datasets 2.19.1. The experiments are conducted on a shared computing cluster with four NVIDIA A100 GPUs.